MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo 461a

February 1978
Revised November 1978

A GLIMPSE OF TRUTH MAINTENANCE

Jon Doyle*

Abstract: To choose their actions, reasoning programs must be able to draw conclusions from limited information and subsequently revise their beliefs when discoveries invalidate previous assumptions. A truth maintenance system is a problem solver subsystem for performing these functions by recording and maintaining the reasons for program beliefs. These recorded reasons are useful in constructing explanations of program actions in "responsible" programs, and in guiding the course of action of a problem solver. This paper describes the structure of a truth maintenance system, methods for encoding control structures in patterns of reasons for beliefs, and the method of dependency-directed backtracking.

*Fannie and John Hertz Foundation Fellow

This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643, and in part by NSF grant MCS77-04828.

Acknowledgements: I thank Gerald Jay Sussman, Richard M. Stallman, Guy L. Steele Jr., Johan de Kleer, Drew McDermott, David McAllester, Scott Fahlman, Howard Shrobe, Charles Rich, Marilyn Matz, Beth Levin, Jim Stansfield, Mitchell Marcus, and Richard Brown for ideas, comments and advice, and the Fannie and John Hertz Foundation for supporting my research with a graduate fellowship.

This paper will appear in the Proceedings of the Fourth Workshop on Automatic Deduction, Austin, Texas, January 1979. An earlier version of the paper will appear in *Artificial Intelligence: a MIT Perspective*, P. H. Winston and R. H. Brown, editors, MIT Press, 1979.

Contents

Introduction

One important problem faced by reasoning programs is the need to make decisions based on limited information. This problem arises both in programs interacting with an external environment and in contemplative programs searching a data base for an answer to some question. There are two consequences of this need to predict; the program must have some way to make decisions based on limited information, and the program must have some way to revise its beliefs if these decisions are found to be in error. The first of these abilities is provided by utilizing epistemic classifications of possible program beliefs so that conclusions may be drawn from the lack of belief as well as from other beliefs. The second ability is in general a very complex problem for which no complete solutions are known. (See Quine and Ullian [1978] and Rescher [1964] for surveys of the problem.) However, the simpler problem of revising beliefs based on limited information is solvable by recording the reasons for each program belief. These records can be used to find the set of extant beliefs by determining which beliefs have valid reasons. These recorded reasons also are useful in resolving conflicts that arise when limited knowledge gives rise to incompatible conclusions. This paper describes a mechanization of these abilities, embodied in a general-purpose problem solver subsystem called a *truth maintenance system.*

A truth maintenance system (TMS) records and maintains "proofs" of program beliefs. It manipulates two data structures; *nodes*, which represent beliefs, and *justifications*, which represent reasons for beliefs. The fundamental actions the TMS can be called upon to perform are the creation of a new node, to which the problem solving program can attach the statement of a belief, and the addition of a new justification to a node, to represent assertion of the belief associated with the node by some rule or procedure in the problem solver. The addition of new justifications may invoke the automatic procedure of *truth maintenance* to make any revisions necessary in the set of beliefs. The TMS revises beliefs by using the recorded justifications to compute non-circular proofs of beliefs from basic hypotheses. These proofs distinguish one or more justifications as the *well-founded support* for each believed node, and are used during truth maintenance to determine the set of beliefs to update by finding those nodes whose well-founded support depends on changed beliefs. These proofs allow another process, *dependency-directed backtracking*, to resolve conflicts by tracing the well-founded supports of conflicting beliefs to remove one of the assumptions causing the conflict and to make a record used to prevents similar future conflicts.

The TMS employs a special type of justification, called a *non-monotonic justification*, to draw conclusions based on limited or incomplete knowledge. This type of justification allows belief in a node to be based not only on other beliefs, as occurs in the standard forms of deduction and reasoning, but also on lack of belief in certain nodes. For example, a node N-1 representing a statement P might be justified on the basis of a lack of belief in a node N-2 representing the belief ~P. (Distinct nodes are used to represent P and ~P.) In this case, the TMS would have N-1 believed as long as N-2 was not believed, and we would call N-1 an *assumption*. (More generally, by assumption we mean any node whose well-founded support is a non-monotonic justification.)

As a small example, suppose an office scheduling program is considering holding a meeting M on Wednesday. To do this, the program assumes that the meeting is on Wednesday. The data base of the program includes a rule which draws the conclusion that due to regular commitments, any meeting on Wednesday must occur at 1:00 PM. However, the fragment of the schedule for the week constructed so far has something else scheduled for that time already, and so another rule in the data base concludes that the day for the meeting cannot be Wednesday. These beliefs might be notated as follows:

Node	*Statement*	*Justification*
N-1	DAY(M) = WEDNESDAY	(SL () (N-2))
N-2	DAY(M) ≠ WEDNESDAY	
N-3	TIME(M) = 13:00	(SL (R-37 N-1) ())

As seen in the above notation for justifications, each justification consists of two lists. The meaning of the notation is that the statement depends on each of the nodes in the first list being believed, and on each of the nodes in the second list not being believed. Since there is no known justification for N-2, it is not believed. The justification for N-1 specifies that it depends on the lack of belief in N-2, and so N-1 is believed. The justification for N-3 shows that it is believed due to rule R-37 acting on N-1. When the assumption N-1 is rejected by some rule,

N-2	DAY(M) ≠ WEDNESDAY	(SL (R-9 N-7 N-8) ())

where N-7 and N-8 represent the day and time of some other engagement, the TMS will revise the beliefs so that N-1 and N-3 are not believed.

Representation of Knowledge about Belief

A node may have several justifications, each of which represents a different reason for belief in the node. The node is believed if at least one of these justifications is *valid*. The conditions for validity of justifications are described below. We say that a node which has a valid justification is *in*, and that a node without a valid justification is *out*. The distinction between *in* and *out* is not that of *true* and *false*. The former classification denotes conditions of knowledge about reasons for belief; the existence or non-existence of valid reasons. *True* and *false*, on the other hand, classify statements according to truth value independent of any reasons for belief. In this way, there can be four states of knowledge about a proposition P, corresponding to the node representing P being *in* or *out* and the node representing ~P being *in* or *out*.

There are two basic forms of justifications. These are inspired by the typical forms of arguments in natural deduction inference systems. A sample proof in such a system might be as follows:

Line	*Statement*	*Justification*	*Dependencies*
1.	A⊃B	Premise	{1}
2.	B⊃C	Premise	{2}
3.	A	Hypothesis	{3}
4.	B	MP 1,3	{1,3}
5.	C	MP 2,4	{1,2,3}
6.	A⊃C	Discharge 3,5	{1,2}

Each step of the proof has a line number, a statement, a justification, and the set of line numbers the statement depends on. Premises and hypotheses depend on themselves, and other lines depend on the set of premises and hypotheses derived from their justifications. The above proof proves A⊃C from the premises A⊃B and B⊃C by hypothesizing A and concluding C. The assumption A is then discharged to provide the proof of A⊃C. There are two effects that justifications can have on the set of dependencies in natural deduction systems; either the justifications can sum the dependencies of the referenced lines (as in line 4), or they can subtract the dependencies of some lines from those of other lines (as in line 6). The two types of justifications used in a TMS account for these effects on dependencies. A *support-list (SL) justification* says that the justified node depends on a set of other nodes, and thus in effect sums the dependencies of the referenced nodes. A *conditional-proof (CP) justification* says that the node it justifies depends on the validity of a certain hypothetical argument, and as in the example above, subtracts the dependencies of some nodes (the hypotheses of the hypothetical argument) from the dependencies of others (the conclusion of the hypothetical argument). These two types of justifications can be used to construct a variety of forms of dependency relationships.

The support-list justification is of the form

(SL <*in*list> <*out*list>).

A SL-justification is valid if each node in its *in*list is *in*, and each node in its *out*list is *out*. A SL-justification can be used to represent several types of deductions. When both the *in*list and *out*list are empty, we say the justification forms a *premise* justification. A premise justification is always valid, and so the node it justifies will always be believed. Normal deductions are represented by support-list justifications with empty *out*lists. These represent monotonic deductions of the justified node from belief in the nodes of the *in*list. *Assumptions* are nodes whose well-founded support is a support-list justification with a nonempty *out*list. These justifications can be interpreted by viewing the nodes of the *in*list as comprising the reasons for making the assumption; the nodes of the *out*list represent the specific incompletenesses of knowledge authorizing the assumption. As will be clear later, it is sometimes convenient to interpret the nodes of the *out*list as assertions implying the negation of the node justified by the justification.

The conditional-proof justification takes the form

(CP <consequent> <*in*hypotheses> <*out*hypotheses>).

A node justified by such a justification represents an implication, whose support is derived by a conditional proof of the consequent node from the hypothesis nodes. A justification of this form is valid if the consequent node is *in* whenever (a) each node of the *in*hypotheses is

in and (b)each node of the *out*hypotheses is *out*. Except in a few esoteric uses, the set of *out*hypotheses is empty. Standard conditional proofs in natural deduction systems specify a single set of hypotheses, which correspond to our *in*hypotheses. The truth maintenance system requires that the set of hypotheses be divided into two disjoint subsets, since nodes may be derived both from some nodes being *in* and other nodes being *out*.

Encoding Control Structures in Sets of Justifications

The use of justifications for recording normal, monotonic deductions is straightforward. Non-monotonic justifications augment the standard deductive relationships between beliefs by allowing the encoding of control structures (patterns of assumptions) into sets of justifications among nodes. The virtue of such an encoding is that the choices underlying problem solver actions become explicit, thus allowing careful failure analysis. In some cases, an automatic procedure like dependency-directed backtracking can perform this failure analysis and set the problem solver onto the next step of its investigation. We describe in detail two important types of control structures; default assumptions and sets of alternatives.

Default Assumptions

One very common technique used in problem solving systems is to specify a default choice for the value of some quantity. This choice is made with the intent of overriding it if either a good reason is found for using some other value, or if making the default choice leads to an inconsistency. The assumption of the day of the week for a meeting in the first example above is such a default assumption.

In the case of a binary choice, a default assumption can be represented by believing a node if the node representing its negation is *out*. When the default is chosen from a set of alternatives, the following generalization of the binary case is used. Let $\{F_1, ..., F_n\}$ be the set of the nodes which represent each of the possible values of the choice. Let G be a node which represents the reason for making the default assumption. Then F_i may be made the default choice by providing it with the justification

$$(\text{SL } (G) \ (F_1 \ldots F_{i-1} F_{i+1} \ldots F_n)).$$

If no information about the choice exists, there will be no reasons for believing any of the alternatives except F_i. Thus F_i will be *in* and each of the other alternatives will be *out*. If some other alternative receives a valid justification from other sources, that alternative will become *in*. This will invalidate the support of F_i, and F_i will become *out*. If a contradiction is derived from F_i, the dependency-directed backtracking mechanism will recognize that F_i is an assumption by means of its dependence on the other alternatives being *out*. (See the section on dependency-directed backtracking for an explanation of this.) The result of backtracking may be to justify one of the other alternatives, say F_j, causing F_i to go *out*. The justification for F_j will be of the form

(SL <various things> <remainders>)

where the remainders are the F_k's remaining after F_i and F_j are taken away. In effect, backtracking will cause the removal of the default choice with the set of alternatives, and will set up a new default assumption structure from the remaining alternatives. As a concrete example, our scheduling program might default a meeting day as follows:

```
N-1     DAY (M) = MONDAY
N-2     DAY (M) = WEDNESDAY          (SL () (N-1 N-3))
N-3     DAY (M) = FRIDAY
```

In this example, Wednesday is assumed to be the day of the meeting M, with Monday and Friday being the alternatives. Wednesday will be the default choice until a valid reason is supplied for either Monday or Friday.

If the complete set of alternatives from which the default assumption is to be chosen cannot be known in advance but must be discovered piecemeal, a slightly different structure is necessary. This ability to extend the set of alternatives is necessary, for example, when the default is a number, due to the large set of possible alternatives. For cases like this the following structure may be used instead. Retaining the above notation, let $\sim F_i$ be a new node which will represent the negation of F_i. We will arrange for F_i to be believed if $\sim F_i$ cannot be proven, and will set up justifications so that if F_j is distinct from F_i, F_j will imply $\sim F_i$. This is done by giving F_i the justification

$$(\mathrm{SL}\ (G)\ (\sim F_i)),$$

and by giving $\sim F_i$ a justification of the form

$$(\mathrm{SL}\ (F_j)\ ())$$

for each alternative F_j distinct from F_i. As before, F_i will be assumed if no reasons for using any other alternative exist. Furthermore, new alternatives can be added to the set simply by giving $\sim F_i$ a new justification corresponding to the new alternative. This structure for default assumptions will behave as did the fixed structure in the case of an unselected alternative receiving independent support. Backtracking, however, has a different effect. If a contradiction is derived from the default assumption supported by the extensible structure, $\sim F_i$ will be justified so as to make F_i become *out*. If this happens, no alternative will be selected to take the place of the default assumption. The extensible structure requires an external mechanism to construct a new default assumption whenever the current default is ruled out. For example, a census program might make assumptions about the number of children in a family as follows:

```
N-1     #-CHILDREN(F) = 2        (SL () (N-2))
N-2     #-CHILDREN(F) ≠ 2        (SL (N-3) ())
                                 (SL (N-4) ())
                                 (SL (N-5) ())
                                 (SL (N-6) ())

N-3     #-CHILDREN(F) = 0
N-4     #-CHILDREN(F) = 1
N-5     #-CHILDREN(F) = 3
N-6     #-CHILDREN(F) = 4
```

With this system of justifications, N-1 would be believed because no different number of children is known. If it turns out that the family has 5 children, a new statement would have to be made, along with a new justification of N-2 in terms of this new statement.

Sets of Alternatives

The default assumption structures allow a choice from a set of alternatives, but do not specify the order in which new alternatives are to be tried if the initial choice is wrong. Such advice sometimes is a linear ordering on the set of alternatives. Linearly ordered sets of alternatives are useful whenever heuristic information is available for making a choice such as the state of a transistor or the day of the week for a meeting.

If it is certain that rejected alternatives are rejected permanently and will never again be believed, the linear ordering on the set of alternatives can be specified by a controlled sequence of default assumptions. This can be implemented by justifying each F_i with

$$(\text{SL } (G \sim F_{i-1}) \ (\sim F_i)),$$

where G is the reason for the set of alternatives. The first alternative F_1 will be selected initially. As each alternatives is ruled out through its negation being justified, the next alternative in the list will be assumed. For example, we might have:

```
N-1     DAY(M) = WEDNESDAY       (SL () (N-2))
N-2     DAY(M) ≠ WEDNESDAY
N-3     DAY(M) = THURSDAY        (SL (N-2) (N-4))
N-4     DAY(M) ≠ THURSDAY
N-5     DAY(M) = TUESDAY         (SL (N-4) ())
```

This would guide the choice of day for the meeting M to Wednesday, Thursday and Tuesday, in that order.

If previously rejected alternatives can be independently rejustified (say by special case rules correcting a choice made by the backtracking system), a more complicated structure is necessary. Such a set of alternatives can be described by the following justifications. For each alternative A_i, three new nodes should be created. These new nodes

are PA_i (meaning "A_i is a possible alternative"), NSA_i (meaning "A_i is not the selected alternative"), and ROA_i (meaning "A_i is a ruled-out alternative"). Each PA_i should be justified with the reason for including A_i in the set of alternatives. Each ROA_i is left unjustified. Each A_i and NSA_i should be given justifications as follows:

A_i: (SL (PA_i NSA_1 ... NSA_{i-1}) (ROA_i))
NSA_i: (SL () (PA_i))
(SL (ROA_i) ())

Here the justification for A_i means that A_i is an alternative, no better alternative is selected, and A_i is not ruled out. The two justifications for NSA_i means that either A_i is not a valid alternative, or that A_i is ruled out. With this structure, processes can independently rule in or rule out an alternative by justifying the appropriate alternative node or ruled-out-alternative node.

This structure is also extensible. New alternatives may be added simply by constructing the appropriate justifications as above. These additions are restricted to appearing at the end of the order. That is, new alternatives cannot be spliced into the linear order between two previously inserted alternatives.

Dependency-Directed Backtracking

Making assumptions admits the possibility of making errors. When a contradiction or other inconsistent state of the data base occurs, the TMS employs a process called dependency-directed backtracking to find and remove incorrect assumptions so as to restore consistency. There are several steps involved in dependency-directed backtracking, but first the inconsistency must somehow be signalled to the TMS, as there is no built-in notion of inconsistency. This signalling consists of informing the TMS that a node represents an inconsistency. With this knowledge, the TMS will try to restore consistency whenever the node comes *in* by rejecting enough assumptions to force the node *out*. Any node may be marked for such treatment. A node so marked is called a *contradiction*.

The steps of dependency-directed backtracking are as follows. First, the well-founded support of the contradiction node is traced backwards to find the set of assumptions (nodes with a non-monotonic justification as their well-founded support) underlying the contradiction. Belief in at least one of these assumptions must be retracted to remove the contradiction. This is done by creating a new justification for one of the *out* nodes underlying one of the assumptions. Since the backtracker may be mistaken in its assignment of blame to that assumption, the justification used to retract the assumption must indicate the alternatives that were available but not utilized by the backtracker. Thus the new justification includes (a) the reason why the contradiction occurred and (b) the other assumptions involved. Thus the second step of the backtracking process is to construct a node recording the reason why the contradiction occurred, and the third step is

to use this node and the other assumptions in justifying an *out* node supporting the assumption selected for removal.

In more detail, the first step of the backtracking process traces backwards through the well-founded support of the contradiction node to collect the set of "maximal" assumptions supporting the contradiction. Not all assumptions found by tracing the well-founded support are used; instead, only those assumptions which do not support other assumptions underlying the contradiction as well. That is, the well-founded support relationships induce a natural partial-ordering on nodes, where one node is said to be "lower" than a second node if the first occurs in the second's well-founded support. The maximal assumptions are then those assumptions which are maximal in this partial order. Only this "front line" of assumptions is used because if the reason for revoking a lower-level assumption involves a higher-level assumption, then the removal of the lower-level assumption would cause truth maintenance to remove the higher-level assumption that it supports, so the reason for removing the lower-level assumption would not hold up. This reflects the fact that there may not be enough information to definitely rule out a lower-level assumption.

The second step summarizes the reason for the contradiction in terms of the set of selected assumptions. Let $S = \{A_1, \ldots, A_n\}$ indicate the set of inconsistent assumptions. The backtracker then creates a node called a *nogood*, a new node signifying that S is inconsistent. Since contradiction nodes really represent the false statement, the nogood node can be taken to represent

$$A_1 \wedge \ldots \wedge A_n \supset false,$$

or alternatively,

(1) $$\sim (A_1 \wedge \ldots \wedge A_n).$$

S is recorded as the *nogood-set* of the nogood. This meaning for the nogood node is produced by justifying it with the conditional proof of the contradiction node relative to the assumption nodes, that is, with the justification

(2) (CP <contradiction node> S ()).

In this way, the inconsistency of the set of assumptions will be remembered even after the contradiction has been resolved by the retraction of some hypothesis.

The final step is selecting an assumption A_i (the "culprit") from S and justifying one of the *out* nodes listed in its well-founded supporting justification. (If these underlying *out* nodes are thought of as "denials" of the assumption, then this step is much like reasoning by *reductio ad absurdum*.) Let NG be the nogood, and let the inconsistent assumptions be $A_1, \ldots, A_n$. Let $D_1, \ldots, D_k$ be the *out* nodes appearing in the justification which supports belief in the assumption A_i. This justification for the assumption can be invalidated by justifying D_1 with the justification

(3) (SL (NG A_1 ... A_{i-1} A_{i+1} ... A_n) (D_2 ... D_k)).

This justification is valid whenever the nogood and other assumptions are believed and the other "denials" of the culprit are not believed. If the choice of culprit was in error, then another contradiction will occur in the future involving D_1, and by this justification will be

led to suspect the remaining assumptions, as well as D_j if there are any other *out* nodes listed in its justification. If, by means of other previously existing justifications, the current contradiction is still *in* following the addition of this justification, backtracking is repeated. Presumably the new invocation of the backtracking process will find that the previous culprit is no longer an assumption. Backtracking halts when the contradiction becomes *out*, or when no assumptions can be found underlying the contradiction.

As an example, consider a program scheduling a meeting, preferably at 10 AM in either room 813 or 801. This might be represented as:

```
N-1     TIME(M) = 1000          (SL () (N-2))
N-2     TIME(M) ≠ 1000
N-3     ROOM(M) = 813           (SL () (N-4)
N-4     ROOM(M) = 801
```

With these justifications, N-1 and N-3 are *in*, and the other two nodes are *out*. If some previously scheduled meeting exists, it might cause this combination of time and room for the meeting to be ruled out by means of a contradiction.

```
N-5     CONTRADICTION           (SL (N-1 N-3) ())
```

The dependency-directed backtracking system then traces the well-founded support of the contradiction to find that it depends on two assumptions, N-1 and N-3, both of which are maximal.

```
N-6     NOGOOD N-1 N-3          (CP N-5 (N-1 N-3) ())
N-4     ROOM(M) = 801           (SL (N-6 N-1) ())
```

A nogood node is created which means, in accordance with form (1) above,

$$\sim(\text{TIME}(M) = 1000 \wedge \text{ROOM}(M) = 813)$$

and this nogood is given a justification corresponding to form (2) above. The assumption N-3 is selected arbitrarily as the culprit, and is rejected by providing its only *out* supporting node, N-4, with a justification of the form (3) above. Following this, N-1, N-4, and N-6 are *in*, and N-2, N-3, and N-5 are *out*. N-6, the nogood node, has an always-valid CP-justification since the contradiction node N-5 depends directly on the two assumptions N-1 and N-3 without any additional beliefs intervening. If some further consideration determines that room 801 cannot be used after all, another contradiction node could be created to force a different choice.

```
N-7     CONTRADICTION           (SL (N-4) ())
N-8     NOGOOD N-1              (CP N-7 (N-1) ())
N-2     TIME(M) ≠ 1000          (SL (N-8) ())
```

Tracing backwards from N-7 through N-4, N-6, and N-1, the backtracker finds that the contradiction depends on only one assumption, N-1. The nogood node N-8 is created and justified with a CP-justification which in effect is equivalent to the SL-justification

(SL (N-6) ()),

since the nogood N-6 contributes to the contradiction but does not itself depend on the assumption N-1. The revocation of the assumption N-1 removes N-5, the previous objection to the choice of room, so at the close of this bit of decision making N-2, N-3, N-6, and N-8 are *in*, and N-1, N-4, N-5, and N-7 are *out*.

There are a number of variations on this particular scheme for dependency-directed backtracking. All of these variations are great improvements over the chronological backtracking systems used in classical systems like MICRO-PLANNER and many early theorem provers. The improvements stem from the non-chronological nature of dependency-directed backtracking, in which the support relationships rather than the temporal orderings determine the choices responsible for an error. Another improvement is that the cause of the contradiction is summarized via a nogood node. This summarization keeps the system from making the same mistake in the future. Stallman and Sussman [1977] have shown that these two improvements lead to enormous gains in efficiency over the chronological systems.

Truth Maintenance Mechanisms

Consider the statements:

F (= (+ X Y) 4)
G (= X 1)
H (= Y 3).

If both *F* and *G* are *in*, then belief in *H* can be justified by

(SL (*F G*) ()).

This justification will cause *H* to become *in*. If *G* subsequently becomes *out* due to changing hypotheses, and if *H* becomes *in* by some other justification, then *G* can be justified by

(SL (*F H*) ()).

Suppose the justification supporting belief in *H* then becomes invalid, thus causing the TMS to reassess the grounds for belief in *H*. If the decision to believe a node is based on a simple evaluation of each of the justifications of the node, then both *G* and *H* will be left *in*, since the two justifications form circular proofs for *G* and *H* in terms of each other. These justifications are mutually satisfactory if *F*, *G* and *H* are *in*. This example points out one of the major concerns in truth maintenance processing; the avoidance of using circular proofs to support beliefs. This is the reason why well-founded support is maintained.

Essentially three different kinds of circularities can arise in purported proofs. The first and most common is a circularity in which all nodes involved can, consistently with

their justifications, be taken to be *out*. Such circularities arise routinely through equivalences and simultaneous constraints, in which many beliefs may be mutually supporting without any of the beliefs having non-circular reasons for being believed. The above algebra example falls into this class of circularity.

The second type of circularity is one in which at least one of the nodes involved must be *in*. An example is that of two nodes *F* and *G*, such that *F* has the justification

(SL () (*G*)),

and *G* has the justification

(SL () (*F*)).

Here either *F* must be *in* and *G out*, or *G* must be *in* and *F out*. This type of circularity arises in defining some sets of alternatives. Frequently other ordered alternative structures can be used to avoid such circularities.

The third form of circularity which can arise is the unsatisfiable circularity. In this type of circularity, no assignment of *in* or *out* to nodes is consistent with their justifications. An example of such a circularity is a node *F* with the justification

(SL () (*F*)).

This justification implies that *F* is *in* if and only if *F* is *out*. Unsatisfiable circularities are bugs, indicating a misorganization of the knowledge of the program using the truth maintenance system. Unsatisfiable circularities are violations of the semantics of *in* and *out*, which can be interpreted as meaning that the lack of reasons for belief in a node is equivalent to the existence of reasons for belief in the node. (It has been my experience that such circularities are most commonly caused by confusing the concepts of *in* and *out* with those of *true* and *false*. For instance, the above example could be produced by this misinterpretation as an attempt to assume belief in the node *F* by giving it the justification (SL () (*F*)).)

In addition to the problems caused by circular proofs, the TMS must also handle problems introduced by conditional-proof justifications. There are two parts to the implemented approach. The validity of CP-justifications is easily checkable only in case the consequent and *in*hypotheses are *in* and the *out*hypotheses are *out*. This is a rare circumstance, however, particularly in the case of backtracking in which a nogood node justified with a CP-justification is used to force *out* the contradiction node appearing as the consequent of the CP-justification. The TMS thus takes the opportunistic and incomplete strategy of using CP-justifications to compute SL-justifications which are equivalent in terms of the dependencies they specify, but which can be checked for validity at any time. Specifically, whenever the CP-justification is found to be valid, an equivalent SL-justification is computed by tracing through the well-founded support of the consequent node of the CP-justification to find the "front line" of nodes which are not in turn supported by any of the *in* or *out*hypotheses. This set of nodes can be divided into the *in* nodes, which form the *in*list of the equivalent SL-justification, and the *out* nodes, which form the *out*list of the equivalent SL-justification. The way these sets of nodes are computed from the well-founded support of the consequent of the CP-justification ensures that the consequent will be in whenever the *in*hypotheses are *in*, the *out*hypotheses are *out*, and the nodes of the equivalent SL-justification respectively *in* and *out*.

The details of the truth maintenance mechanisms will not be pursued here. Many details, along with an annotated implementation, are presented in [Doyle 1978].

Discussion

Truth maintenance systems solve part of the belief revision problem, and provide an associated mechanism for making assumptions based on limited information. It has long been recognized that making assumptions is a necessary part of AI systems, and many systems have employed some mechanism for this purpose. (For example, [Bobrow and Winograd 1977, de Kleer et al. 1977, 1978, Hayes 1973, 1977, Hewitt 1972, Joshi and Rosenschein 1975, McCarthy 1977, McCarthy and Hayes 1969, McDermott 1974, Minsky 1962, 1975, Reiter 1978, Roberts and Goldstein 1977, Sandewall 1972, Sussman et al. 1971].) Unfortunately, the related problem of belief revision received somewhat less study. Most work on revising beliefs was done in the framework of backtracking algorithms operating on rather simple systems of states and actions. The more general problem of revising beliefs based on records of deductions has only been examined in more recent work. (See [Cox and Pietrzykowski 1976, Doyle 1978, Fikes 1975, Hayes 1975, Katz and Manna 1976, Latombe 1977, London 1978, McAllester 1978, McDermott 1974, 1977, Nevins 1974, Srinivasan 1976, Stallman and Sussman 1977].) The literature of philosophy and logic contains a large amount of work on the belief revision problem (see [Quine and Ullian 1978, Rescher 1964]), as well as some work on formal methods for making decisions based on limited information. The history of attempts at formalizing the AI methods for making assumptions is surveyed by McDermott and Doyle [1978], who also present a mathematical semantics for what is termed *non-monotonic logic*.

Truth maintenance systems lend themselves to other uses as well as belief revision and making assumptions. Generating explanations is an immediate application. The recorded reasons for beliefs can form the basis of an explanation system in "responsible" programs [Sussman, personal communication] which can justify their actions and beliefs to a user. A crucial aspect of the problem of explanation is that unless care is taken in structuring the knowledge of the program, the explanations will contain information at many levels of detail, thus making the explanation incomprehensible. It is thus important to try to structure the use of a truth maintenance system so that levels of detail in explanations are separated automatically. Doyle [1978] describes a method by which conditional proofs are used to factor unwanted low-level details from explanations. When such factoring is done at each level, a hierarchical structure emerges in explanations.

Truth maintenance systems can be applied to the problem of controlling problem solvers in several ways. The simplest method is that of using an automatic procedure like dependency-directed backtracking for guiding the search. More sophisticated methods can be designed which represent control decisions as explicit program beliefs, and separate the reasons for control decisions from the reasons for beliefs derived in response to the control decisions. With such a separation, careful failure and choice analysis routines can examine the history of the problem solver, and much information can be salvaged from mistakes. (See [de Kleer et al. 1977, Doyle forthcoming, Stallman and Sussman 1977].)

References

Bobrow, D. G. and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cognitive Science*, Vol. 1, No. 1, 1977.

Cox, P. T. and T. Pietrzykowski, "A Graphical Deduction System," Department of Computer Science Research Report CS-75-35, University of Waterloo, July 1976.

de Kleer, J., J. Doyle, C. Rich, G. L. Steele Jr. and G. J. Sussman, "AMORD: A Deductive Procedure System," MIT AI Lab, Memo 435, January 1978.

de Kleer, J., J. Doyle, G. L. Steele Jr. and G. J. Sussman, "Explicit Control of Reasoning," MIT AI Lab, Memo 427, June 1977, also *Proc. ACM Symp. on Artificial Intelligence and Programming Languages*, Rochester, New York, August 1977.

Doyle, J., "Truth Maintenance Systems for Problem Solving," MIT AI Lab, TR-419, January 1978.

Doyle, J., "Reflexive Interpreters", forthcoming.

Fikes, R. E., "Deductive Retrieval Mechanisms for State Description Models," *Proc. Fourth International Joint Conference on Artificial Intelligence*, September 1975, pp. 99-106.

Hayes, P. J., "The Frame Problem and Related Problems in Artificial Intelligence," in A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, San Francisco: Josey-Bass, 1973.

Hayes, P. J., "The Logic of Frames", University of Essex, November 1977.

Hayes, P. J., "A Representation for Robot Plans," *Proc. Fourth IJCAI*, September 1975, pp. 181-188.

Hewitt, C. E., "Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot," MIT AI Laboratory TR-258, 1972.

Joshi, A. K. and S. J. Rosenschein, "A formalism for relating lexical and pragmatic information: its relevance to recognition and generation," *Proc. Workshop on Theoretical Issues in Natural Language Processing*, Cambridge, Massachusetts, 1975, pp. 79-83.

Katz, S. and Z. Manna, "Logical Analysis of Programs," *Comm. ACM*, Vol. 19, No. 4, pp. 188-206.

Latombe, J.-C., "Une Application de l'intelligence Artificielle a la Conception Assistee par Ordinateur (TROPIC)," Universite Scientifique et Medicale de Grenoble, thesis D.Sc. Mathematiques, November 1977.

London, P. E., "Dependency Networks as a Representation for Modelling in General Problem Solvers," Department of Computer Science TR-698, University of Maryland, September 1978.

McAllester, D. A., "A Three-Valued Truth Maintenance System", MIT AI Lab, Memo 473, May 1978.

McCarthy, J. and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in B. Meltzer and D. Michie, *Machine Intelligence 4*, New York: American Elsevier 1969, pp. 463-502.

McCarthy, J., "Epistemological Problems of Artificial Intelligence," *Proc. Fifth IJCAI*, pp. 1038-1044, 1977.

McDermott, D., "Assimilation of New Information by a Natural Language-Understanding System," MIT AI Lab, AI-TR-291, February 1974.

McDermott, D., "Flexibility and Efficiency in a Computer Program for Designing Circuits," MIT AI Lab, TR-402, June 1977.

McDermott, D. and J. Doyle, "Non-Monotonic Logic I", MIT AI Lab, Memo 486, August 1978.

Minsky, M., "Problems of Formulation for Artificial Intelligence," *Proc. Symp. on Mathematical Problems in Biology*, American Mathematical Society, Providence, RI, 1962, pp. 35-46.

Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. H. Winston, ed., New York: McGraw-Hill 1975, pp. 211-277.

Nevins, A. J., "A Human-Oriented Logic for Automatic Theorem Proving," *J. ACM 21*, #4 (October 1974), pp. 606-621.

Quine, W. V. and J. S. Ullian, *The Web of Belief*, second edition, New York: Random House, 1978.

Reiter, R., "On Reasoning by Default," *Proc. Second Symp. on Theoretical Issues in Natural Language Processing*, Urbana, Illinois, August 1978.

Rescher, N., *Hypothetical Reasoning*, Amsterdam: North Holland 1964.

Roberts, R. B. and I. P. Goldstein, "The FRL Manual," MIT AI Lab AI Memo 409, September 1977.

Sandewall, E., "An Approach to the Frame Problem, and its Implementation," *Machine Intelligence 7*, pp. 195-204, 1972.

Srinivasan, C. V., "The Architecture of Coherent Information System: A General Problem Solving System," *IEEE Transactions on Computers*, Vol. C-25, No. 4, April 1976, pp. 390-402.

Stallman, R. M. and G. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, Vol. 9, No. 2, (October 1977), pp. 135-196.

Sussman, G. J., T. Winograd and E. Charniak, "MICRO-PLANNER Reference Manual," MIT AI Lab, AI Memo 203a, December 1971.